# **BLOCKS OS**

# **An Engineering Project in Community Service**

## Phase – II Report

**Submitted by**

|  |  |  |
| --- | --- | --- |
| **SERIAL NO** | **NAME** | **REGISTRATION NUMBER** |
| 1 | V Surya Kumar | 19BCE10286 |
| 2 | Abhishek Srivastava | 19BCE10071 |
| 3 | Anjali Singh | 19BCG10003 |
| 4 | Pratul Maurya | 19BCY10036 |
| 5 | Pravir Kadian | 19BCE10006 |
| 6 | Saransh Pratap Singh | 19BCY10035 |
| 7 | Viplav Khubchandani | 19BAI10106 |
| 8 | C S Soujanya Mudliar | 19BCG10094 |

***in partial fulfilment of the requirements for the degree of***

***Bachelor of Engineering and Technology***

****

**VIT Bhopal University**

**Bhopal**

**Madhya Pradesh**

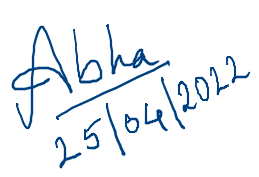
**21 April 2022**

****

**Bonafide Certificate**

Certified that this project report titled **“BLOCKS OS”** is the bonafide work of **19BCE10286 V Surya Kumar, 19BCE10071 Abhishek Srivastava, 19BCE10006 Pravir Kadian, 19BCG10003 Anjali Singh, 19BCY10036 Pratul Maurya, 19BCY10035 Saransh Pratap Singh, 19BAI10106** **Viplav Khubchandani, 19BCG10094 C.S. Soujanya Mudliar** who carried out the project work under my supervision.

This project report (Phase II) is submitted for the Project Viva-Voce examination held on 21st April 2022.



Dr. Abha Trivedi

**Supervisor**

**Comments & Signature (Reviewer 1)**

**Comments & Signature (Reviewer 2)**

# **INTRODUCTION**

Blocks-OS is an open-source operating system built from the ground up for educational purposes and to learn how an operating system works. End-users can use this project to learn more about how computers work by seeing how concepts like interrupt vectors, memory management, and segmentation are implemented on their own computers.

A project like this has several major advantages over a higher-level project that isolates students from the machine:

* Students get a better grasp of the computer by seeing concepts like segmentation, interrupt vectors, and memory management in action.
* The ability to test their operating systems on their own machine helps to make the often abstract notions more concrete.
* Using the project as boiler-plate code and thorough documentation for reference, students can pick any branch of the entire project and begin creating rest features or any individual feature.
* When students are forced to programme without the usual library services, they discover how much help they can obtain from operating systems.

## **1.1 Motivation**

Operating systems are created for a variety of purposes. Although each developer may have their own motives, some developers share the following:

* Students are fond of using existing operating systems, without even noticing the support it is giving. Developing the very beginning part of the software which runs right after we switch on the pc helps us gain a deeper understanding of the system.
* Controlling the machine completely. When creating an application or other userspace programme, the developer must consider code written by others, such as the operating system, libraries, and other applications. It's a tremendous feeling to know that your code is the only one running on a machine.
* The domain of Operating systems includes a lot of research because no detailed guide or course is available covering the implementation of this vast subject. Which improves the opportunities for research. Where studies are carried out to improve existing operating systems.
* Because one must do everything, low-level programming is a fun and exciting task. This may appear to be more difficult, but it is also more enjoyable for the same reasons.

As software developers, we should understand how systems work, how everything fits together, and the inner workings of your programme.

## **Objective**

The operating system curriculum currently available at various universities emphasises the theoretical aspect of the subject. Our project will help the student developer community learn about the practical aspects of the subject by documenting each step.

BLOCK OS differs from other existing projects in that they are mostly incomplete, poorly documented, and lack community support. Our project is created exclusively for students to understand and follow the general process of how an operating system is constructed from the ground up, what goes on behind the scenes of how operating systems work, and much more.

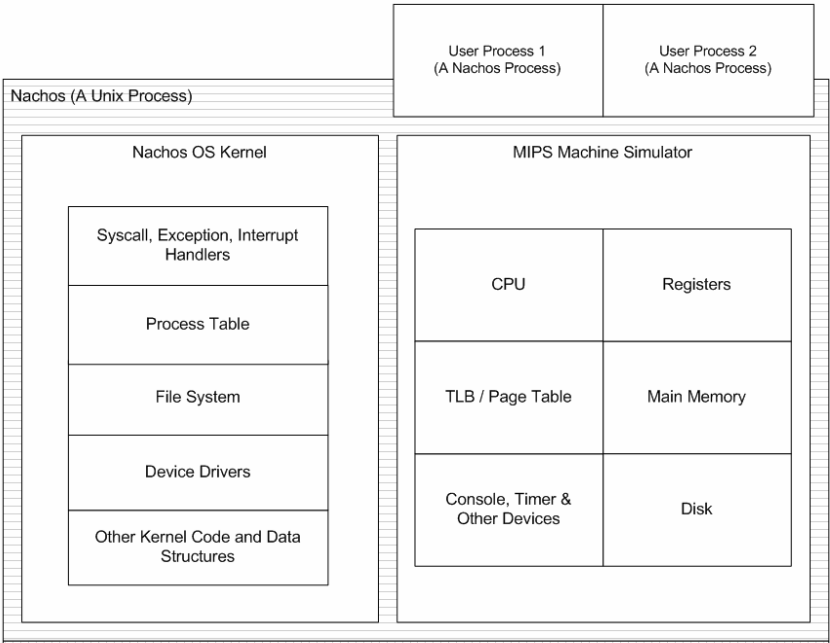
In order to design an OS that any learner can follow a lot of documentation that's well-organized and sequenced so that anyone who reads it can comprehend it quickly. Both the documentation and the implementation will be clearly defined and divided into modules.

# **Existing Work / Literature Review**

On the internet, there are various articles and projects on operating systems, but in order to work on any OS-related project, you must first understand what makes them special, what they perform, and why they follow the architecture they do.

NachOS is an example of an operating system produced by the developer community or by developers, and it is used to implement instructional operating systems. It's a standard UNIX process. Threads and remote procedure calls, as well as current hardware advancements like RISCs and the predominance of memory structures, make implementing functions simply. It also uses the protocol, which is a prominent design strategy.

Furthermore, advanced software and design approaches such as protocol layering and object-oriented programming are available.



**NACH-OS ARCHITECTURE**

PintOS, whose structure and shape are inspired by the NachOS educational operating system from the University of California, Berkeley, is another example of an operating system. PintOS is a simple operating system framework for the 80x86 architecture, written in C like other operating systems. At a very basic level, it implements several capabilities such as kernel threads support, loading and running user programmes, and a file system. PintOS can be run in a system simulator with ease.

PintOS began as a replacement for NachOS, which had a striking visual resemblance to it. PintOS differs from NachOS in two important ways. PintOS, for example, runs on real or simulated 80x86 hardware, whereas NachOS runs as a process on the host operating system. Second, PintOS is developed in C++, whereas NachOS is written in C, unlike most real-world operating systems.

Wiki OS dev is a website with 693 wiki articles that gives information on the creation of operating systems and serves as a community for those interested in OS development. However, when it comes to real implementation, it's a little difficult to rely solely on this because it's more of compacted documentation with an overview of 1-2 functionalities of each.

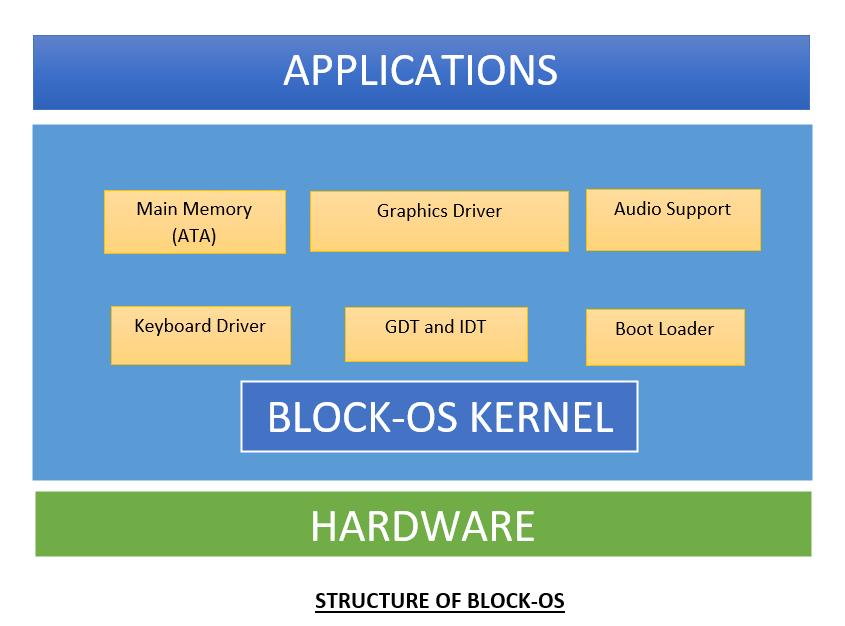
After reviewing and studying many more examples, we came up with the idea of building our operating system with an initial design derived from pre-existing operating system architecture and the addition of some additional functionalities with the goal of learning and understanding how operating systems work and are implemented from the ground up.

# **The topic of the work**

**a) System Design / Architecture (Monolithic OS)**

The monolithic operating system is a very basic operating system in which file management, memory management, device management, and process management are directly controlled within the kernel. The kernel will access all the resources given within the system. In monolithic systems, every element of the software package is contained inside the kernel.​

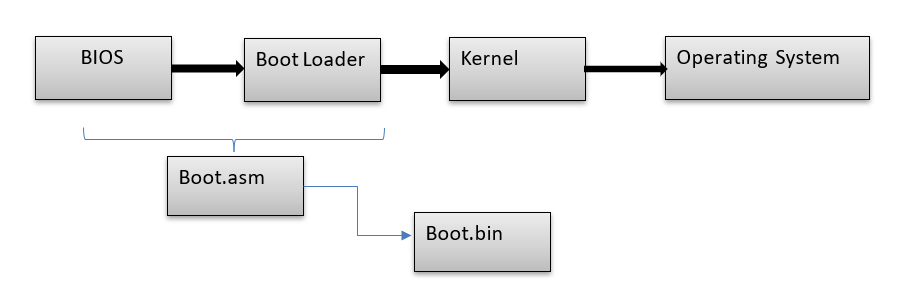
A monolithic kernel has two parts kernel space and user space, both parts communicate with each other through IPC.​

****

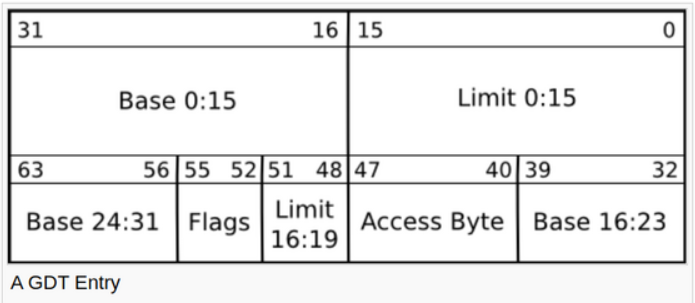
Some of the advantages of the monolithic kernel are −

* The execution of the monolithic kernel is quite fast as the services such as memory management, file management, process scheduling etc. are implemented under the same address space.
* A process runs completely in single address space in the monolithic kernel.
* The monolithic kernel is a static single binary file.

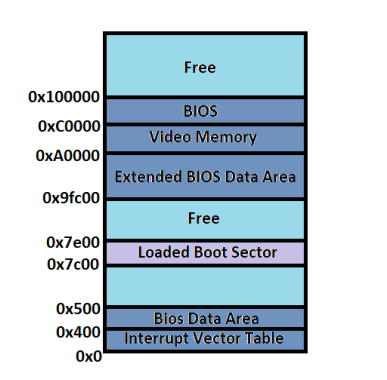
The boot sector is a crucial component of every operating system. The boot sector is implemented in the boot.asm file, which generates the boot.bin file when run. The following figure depicts the overall connection between the BIOS and the boot sector:



With the help of GDT, we were able to develop a protected mode (Global Descriptor Table). In Protected mode, GDT is used to implement each segment using the Flat memory model. The following is the structure of the GDT entry:

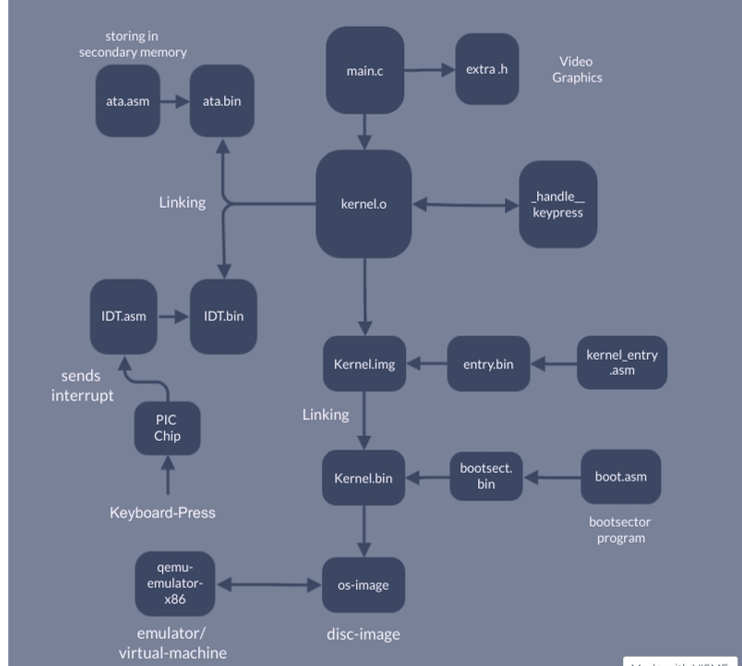


The code and other resources are present in the memory as shown in the below diagram:



**b) Working Principle**

The Block diagram of all the connected files in work is shown in the below diagram:

****

**Protected Mode:**

32-bit operation mode that allows us to use memory up to 4GB.​ Easy implementation and allows to code in C.​ Define an area of memory that is directly accessible by user mode and that is not using GDT(Global Descriptor Table).

**Global Descriptor Table:**

Each segment in a protected mode has a set of properties and defined location, which is defined in GDT where descriptor- lists all properties of a segment.​

​

We have implemented GDT using the Flat memory model which is a single contiguous space model.​

**Audio**

A speaker can have positions "in" and "out". A setting of 1 directs the speaker to move to the "out" position and a setting of 0 directs the speaker to move to the "in" position. Moving in and out repeatedly produces audible tones if the speed of repetition is within the range that the speaker can produce and that the human ear can hear.​

We have implemented three functions in extra.h file that are:​

* **void scream(int fr);** // passes frequency of sound to be played​
* **void play(unsigned int fr );** // plays the sound​
* **void stop( );** //stops the sound​

When a user enters the PLAY( ) command then the audio gets played.​

​

​

**Video Graphics: Graphics Driver​**

We have implemented various printf commands and colour change functions using a graphic driver that includes the cls function which is used to clear the screen ​set Monitor colour which sets the colour of the monitor screen. The print string function is used to print string values on the screen by passing it with an address of a null-terminated character array. ​ Whereas print char prints character values on the screen ​

​

The scroll feature is implemented via the scroll function. We don't have to use the scroll() function ourselves. The printing functions will take care of it.​ The print colour string function accepts two arguments and prints a string on the screen with a specific background and foreground colour.​

​

The print colour char function prints char values of a specific background and foreground colour on the screen in a similar way​

**Secondary Memory​**

Hard Disk is our Primary "Secondary Storage Device". When implementing a hard disk driver, we usually use the in and out assembly commands. There are two types of Hard disk HDD and SSD. For our OS, we are using HDD.

We implemented a hard disk drive for the ATA technology. ATA or Advanced Technology Attachment allows hard disks and CD-ROMs to be internally connected to the motherboard and perform input/output functions.​

There are different ways to read/write to a hard disk. What we have used is (Linear Block Address) LBA mode. This is the easiest way to read/write to a hard disk, all we need to do is pass the Block address of the sector. ​Passing 0 will give us access to the first sector(The boot sector). Please note not to write to the 0’th sector as this can make your computer non-bootable, but you could always copy a boot loader to that sector.​

**Keyboard Drive**

Whenever a key is being pressed, The CPU will call a function that we predefine for it. The processor won’t do the keyboard logic, but it will let us execute some code whenever a key is being pressed.​ We need to make some code to handle the keyboard input, pass its address to the interrupt descriptor table and say to the processor to load it.​

After loading all the parameters and the location of our keyboard handling code, It will call that code whenever a key is being pressed.​ When we get this interrupt, We could try reading from the keyboard which gives us the key being pressed. ​

**Global Constructors​**

The global constructors are such as those on global C++ objects which are supposed to have run before the main function.​

The constructors have the responsibility of parsing the command line arguments, initializing the standard library (memory allocation, signals, ...), running the global constructors and finally exit(main(argc, argv)).​

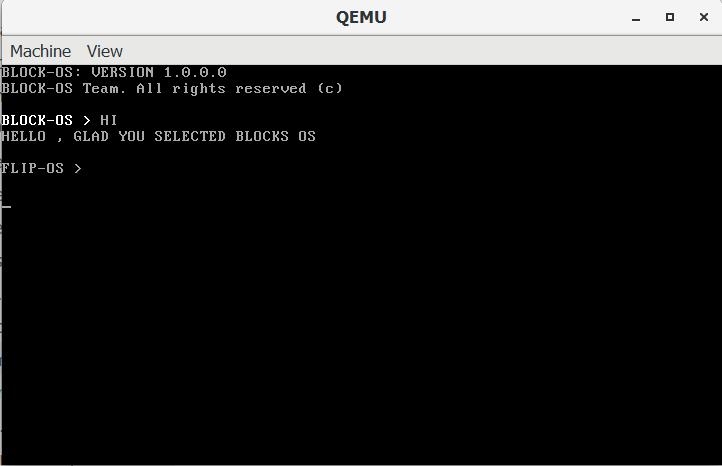
Which should be done before invoking the main method.​ The global constructors/destructors are stored in a sorted array of function pointers and invoking these is as simple as traversing the array and running each element.​

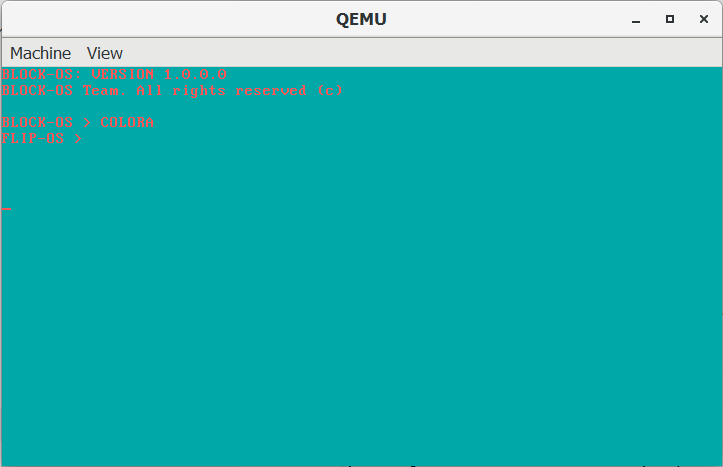
​

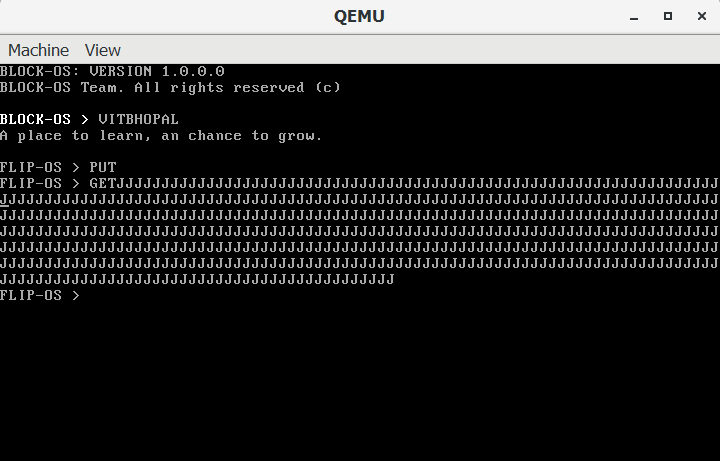
**c) Results and Discussion**

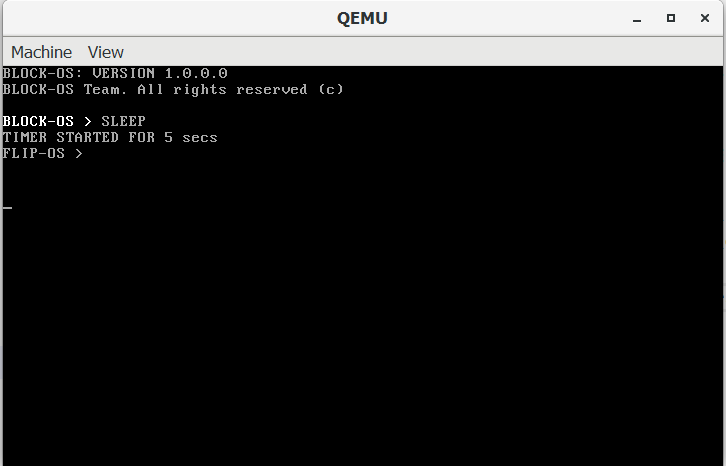
As a result we were able to implement the following features in our BLOCK-OS that are:

* Set up the BIOS and our kernel at the initial level.
* Boot Sector and Boot Loader for booting process.
* Implementation of GDT(Global descriptor table).
* Switch to 32-bit protected mode from 16-bit mode.
* Added support for C.
* Implemented a Graphics Driver that displays graphics on the screen.
* Added functions like COLORA (for changing screen monitor colour), VITBHOPAL(to display messages on the screen) etc.
* Implemented support for calling global constructors which is highly important as we move further with the development.
* We have been able to create delays (Programmable Interval Timer), by creating a sleep function that results in controllable delays.
* Implemented ISR (Interrupt Descriptor Table).
* Set up a Keyboard driver using ISR and PIC.
* Hard disk writing and reading function using ATA.
* Set up audio support using the PLAY() command.









# **4. Individual Contributions**

**V Surya Kumar 19BCE10286**​

* I have explored various OS modules and developed the build script which compiles and assembles the source code to object files and links them to create binary executables.
* I have developed the keyboard driver and respective assembly module, which loads the IDT into the memory and enables the system to take interrupts from the keyboard.
* I have implemented an Interrupt service routine for the keyboard to handle the interrupts generated by the PIC chip and to read and store the scan codes from the console into the main memory.
* I have worked on implementing a programmable interval timer, which generates controllable delays.
* I have also researched and implemented the method of calling global constructors, which will help the project in the long run to provide support for the development of large modules.

# **CONCLUSION**

The basic idea behind the Block OS project is to create an OS for the student community and to showcase how an OS is developed step by step.

The main method that is followed while creating this Operating system is the software development waterfall model method which involves a sequential development process that flows like a waterfall through all processes of a project (like, analysis, design, development, and testing).

This operating system project gives student developers in-depth knowledge of operating systems, which is a key subject for computer science majors. Further, based on the feasibility of the time and development process, various modules can be introduced for future development.

The testing of the Operating system is done on

* Qemu
* Virtual machines
  + VM ware fusion tech (version. 21H1)
  + Virtual Box (version 5.2.32)
* Cloud OS, Amazon AWS Lightsail

# **Reference:**

* <https://wiki.osdev.org/Main_Page>
* <https://github.com/cfenollosa/os-tutorial>
* <https://drive.google.com/file/d/1bUAbfE7OU6NjnyFwVGGkeHR11BPq1l32/view>
* <https://www.cs.bham.ac.uk/~exr/lectures/opsys/10_11/lectures/os-dev.pdf>
* [CSE 221 - Graduate Operating Systems](https://cseweb.ucsd.edu/classes/sp00/cse221/projects.html)
* <https://web.stanford.edu/class/cs140/projects/pintos/pintos_1.html>
* <https://cseweb.ucsd.edu/classes/sp09/cse120/projects/nachos.pdf>
* <https://littleosbook.github.io/>
* <https://student.cs.uwaterloo.ca/~cs350/common/os_overview.html>

**PLAGIARISM REPORT:**

(Report is in 2 parts due to word limit 2000 in plagiarism checker).

<https://drive.google.com/drive/folders/1WueTksS3rgMaA1pCDM3OK2S7bBeRN710?usp=sharing>